Minimal Forth

Peter Knaggs Paul E. Bennett

September 27, 2015

Now that the Forth 2012 document has been published, it is time to review the direction on the standards effort. Both the '94 standard and the '12 document are directed at the professional Forth programmer. Providing a set of expectations that allows both the programmer and the program to be portable between different standard systems.

We argue that it is time to consider the niche market where Forth is generally used, that is embedded control systems. Such systems are often small will little memory and no or limited user Input/Output. The Core word set includes many words that are not relevant to such systems (around 100 words). Such embedded control systems require a small number of words.

However, even a minimal Forth system would tend to provide a full implementation of both the Core and Core-Ext word sets.

The current standard has separated into several sections within the same document, some of the words that one would expect in a minimal system are spread across several different word-sets. To this end, we would like to start a discussion over what would be the minimal word-set we would expect of any Forth. The aims of this discussion is two-fold:

First, from a professional standpoint for those developing critical controls, having a small system footprint that can be fully verified and validated is beneficial. The workload in knowing you have a good basis from which to springboard the application is reasonably simple as it will not take a long time to perform certification confirmation efforts. You can only certify from a know surface (surfaces are the interface between lower level and upper level words).

Second, from an educational standpoint, a Forth with 400+ words is rather daunting for a those wanting to learn a new language. Even knowing that they do not have to do so all at once still leaves many confused. Therefore, a less cluttered dictionary at the start will help the beginners learning process. This will be especially the case for someone who has never programmed before. Table 1 is a comparison of items to lean for different languages. It shows Forth as being well out it front with around 100 more items to lean than other languages, and that assumes the student only looks at the core word set.

Both of these aims could be met with a minimal Forth word set provided there is a reasonable number of useful words to aid in application creation and debugging. The only question then becomes what words and how many?

Frank Sergeant proposed a Forth with just 3 words (`XC@`, `XC!`, and `XCALL`). However, that is only useful in umbilical development and probably only by those who are already well experienced with creating such systems. Table 1 shows us that keeping a minimal word set to around 70–80 words would be acceptable from an educational standpoint. Staying away from desk-top aspects, and just sticking to controllers, it is expected around 50 would be reasonable for use on smaller controllers and provide a capable enough basis to grow control applications. At this end of consideration we are dealing with controllers the likes of the MSP430 and low end ARM systems used for embedded control applications and robotics. Such systems do not tend to have disk or graphics screens but will usually have a low-tech

Langauge	Words	keywords	operators	functions
Forth ’79	129			
Forth ’83	131			
Forth ’12	182	133	49	450
		(core)	(core-ext)	(overall)
Minimal Forth	69			
C	96	32	39	25
Java	85	50	39	—
C#	116	77	39	—
Ada	85	73	12	—

Table 1: Number of “words” in different languages

terminal communications capability and enough resource to run a minimal Forth system.

Proposal

We propose the standard be reduced to a basic description of the abstract machine that represents the language, complete with a limited minimal word set. The composition of this word set is a topic of debate, and appendix A gives a list of 69 words that we propose as a starting point for that debate.

Such a language description would allow people to grasp the language without being daunted by the size of the language. It would also allow for a possible formal description of the language, leading to certified compilers, and thus confidence in the program code.

In the description of the abstract machine it may be useful to allow for a number of more advanced topics. However, it would not be necessary to include words to access these functions in the minimal word set. Such topics could include:

- exception handling
- interpretation
- Multitasking/threading

The words necessary to access these features can be given in a number of supplements to the base description. These supplements may extend the abstract machine, or simply provide additional word sets.

A number of proposed supplements include, but are by no means limited to, the following:

- IEEE Floating Point
- String
- Double numbers
- Local variables
- Heap / User memory
- File access
- Networking / Sockets
- Internationalisation
- Multitasking (cooperate)
- Multitasking (pre-emptive)
- Security (cryptography)
- Exceptions

A Proposed minimal word set

1 Memory Access

6.1.0010	`!`	store	6.1.0850	`C!`	c-store
6.1.0150	`,`	comma	6.1.0860	`C,`	c-comma
6.1.0650	`@`	fetch	6.1.0870	`C@`	c-fetch
6.1.0705	`ALIGN`			`CALIGN`	c-allign
6.1.0706	`ALIGNED`			`CALIGNED`	c-alligned
6.1.0880	`CELL+`	cell-plus	6.1.0897	`CHAR+`	char-plus
6.1.0890	`CELLS`		6.1.0898	`CHARS`	chars

2 Arithmetic

6.1.0120	`+`	plus	6.1.0160	`-`	minus
6.1.0090	`*`	star	6.1.0230	`/`	slash
6.1.0320	`2*`	two-star	6.1.0330	`2/`	two-slash
6.1.0110	`*/MOD`	star-slash-mod	6.1.1890	`MOD`	

3 Logic

6.1.0270	`0=`	zero-equals	6.1.0530	`=`	equals
6.1.0480	`<`	less-than	6.1.0540	`>`	greater-than
6.1.0720	`AND`		6.1.1980	`OR`	
6.1.1720	`INVERT`		6.1.2490	`XOR`	x-or
6.2.2298	`TRUE`		6.2.1485	`FALSE`	
6.1.1805	`LSHIFT`	l-shift	6.1.2162	`RSHIFT`	r-shift

4 Stack

6.1.1290	`DUP`	dupe	6.1.1260	`DROP`	
6.1.2260	`SWAP`		6.1.1990	`OVER`	
6.1.0580	`>R`	to-r	6.1.2060	`R>`	r-from
6.1.2070	`R@`	r-fetch	6.1.2160	`ROT`	rote

5 Flow Control

6.1.1700	`IF`		6.1.1310	`ELSE`	
6.1.2270	`THEN`		6.1.0760	`BEGIN`	
6.1.2430	`WHILE`		6.2.0700	`AGAIN`	
6.1.2140	`REPEAT`		6.1.2390	`UNTIL`	
6.1.1240	`DO`		6.1.1800	`LOOP`	
6.1.1680	`I`		6.1.1730	`J`	
6.1.0070	`'`	tick	6.1.1370	`EXECUTE`	

6 Definitions

6.1.0450	`:`	colon	6.1.0460	`;`	semicolon
6.1.0950	`CONSTANT`		6.1.2410	`VARIABLE`	
6.1.1000	`CREATE`		6.1.1250	`DOES>`	does

7 Device

6.1.1750	`KEY`		10.6.1.1755	`KEY?`	key-question
6.1.1320	`EMIT`		6.1.0990	`CR`	c-r

8 Tools

6.1.0080	`(`	paren	6.2.2535	`\`	backslash
15.6.1.0220	`.S`	dot-s			